CartoonGAN: Generative Adversarial Networks for Photo Cartoonization

Elective in Artificial Intelligence

Giulia Cassara', Ivan Colantoni, Flavio Lorenzi

SAPIENZA
UNIVERSITÀ DI ROMA

Contents

Authors

CARTOON GAN: Generative Adversarial Networks for Photo Cartoonization

Yang Chen
Tsinghua University, China
chenyang15@mails.tsinghua.edu.cn

Yu-Kun Lai
Cardiff University, UK
Yukun.Lai@cs.cf.ac.uk

Yong-Jin Liu*
Tsinghua University, China
liuyongjin@tsinghua.edu.cn

Introduction

Their solution belongs to learning based methods,which have recently become popular to stylize images in artistic forms such as paintings...

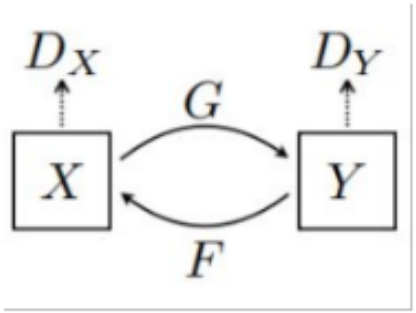
VS

[6] L. Gatys, A. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[11] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *ACM SIGGRAPH*, pages 327–340, 1998.

[12] S.-S. Huang, G.-X. Zhang, Y.-K. Lai, J. Kopf, D. Cohen-Or, and S.-M. Hu. Parametric meta-filter modeling from a single example pair. *The Visual Computer*, 30(6-8):673–684.

[38] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision*, 2017.

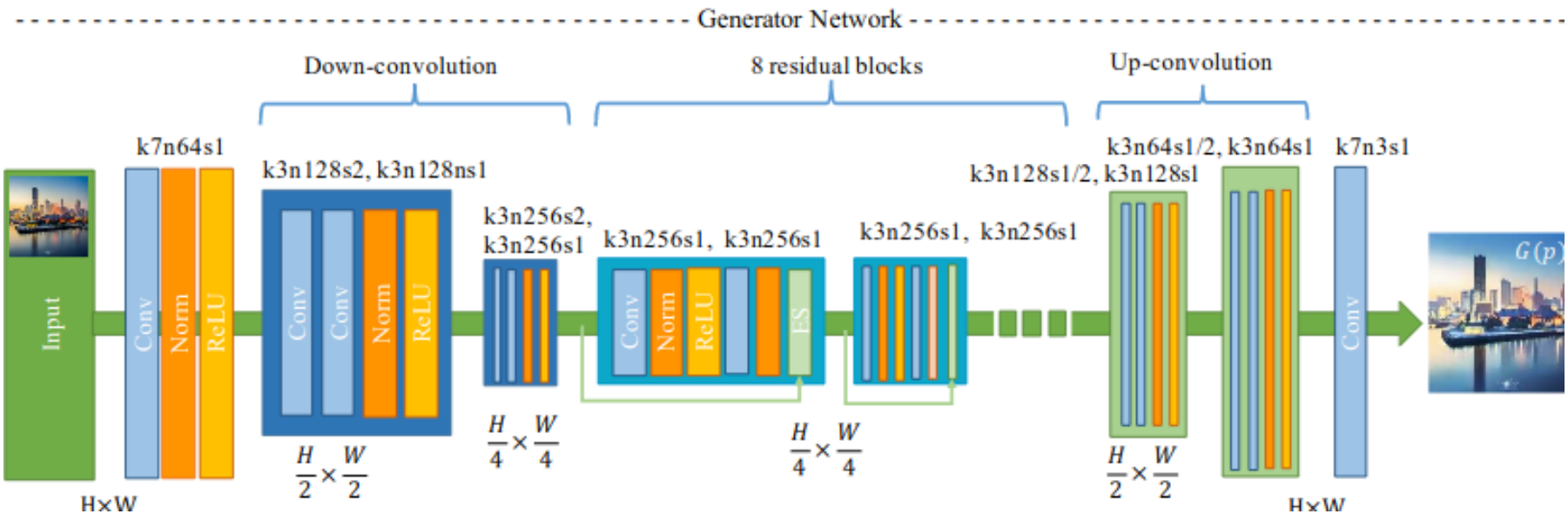


and many others!

Contents

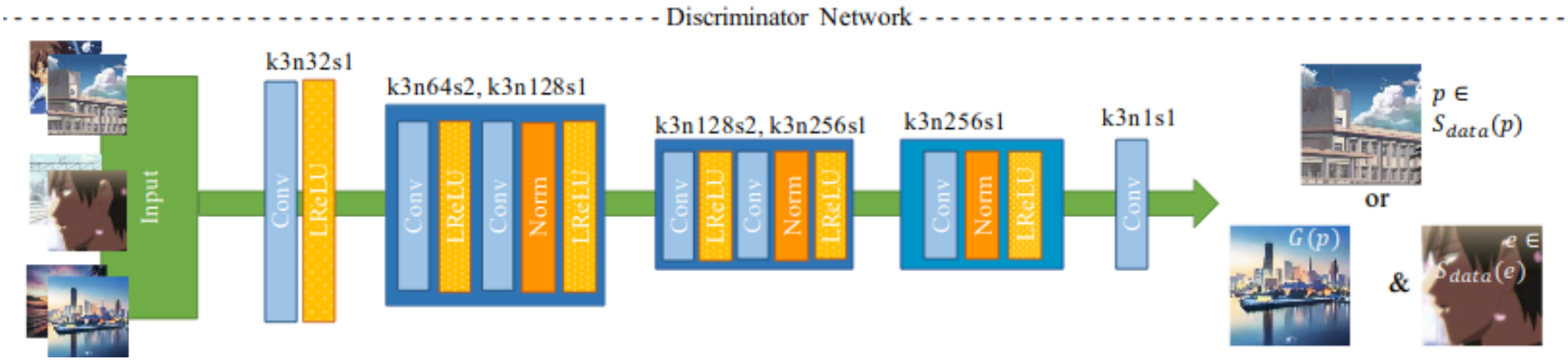
Architecture-Generator

$$(G^*, D^*) = \arg\min_G \max_D \mathcal{L}(G, D)$$



CARTOONGAN architecture: k = kernel size, n = number of feature maps, s = stride in each layer. Norm indicates a normalization layer and 'ES' the elementwise sum

Architecture-Discriminator



- reduced resolution for encoding essential local features
- feature construction block
- 3x3 convolutional layer

Loss Functions

$$\mathcal{L}(G, D) = \mathcal{L}_{adv}(G, D) + \omega \mathcal{L}_{con}(G, D)$$

$$\begin{aligned}\mathcal{L}_{adv}(G, D) = {} & \mathbb{E}_{c_i \sim S_{data}(c)}[\log D(c_i)] \\ & + \mathbb{E}_{e_j \sim S_{data}(e)}[\log(1 - D(e_j))] \\ & + \mathbb{E}_{p_k \sim S_{data}(p)}[\log(1 - D(G(p_k)))]\end{aligned}$$

$$\mathcal{L}_{con}(G, D) = \mathbb{E}_{p_i \sim S_{data}(p)}[||VGG_l(G(p_i)) - VGG_l(p_i)||$$

(a) A cartoon image c_i

(b) The edge-smoothed version e_i

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Initialization Phase

Since the GAN model is highly non-linear, with random initialization, the optimization can be easily trapped at sub-optimal local minimum..

(a) Original photo

(b) Image after initialization

This initialization phase improve fast convergence to a good configuration

Contents

Dataset

Photos. +5.000 real-world photos are downloaded from Torrent.

Cartoon images. We captured key frames from the Cartoon film "Spirited Away" of Hayao Miyazaki.

Figure: Example of our dataset

Smoothed images

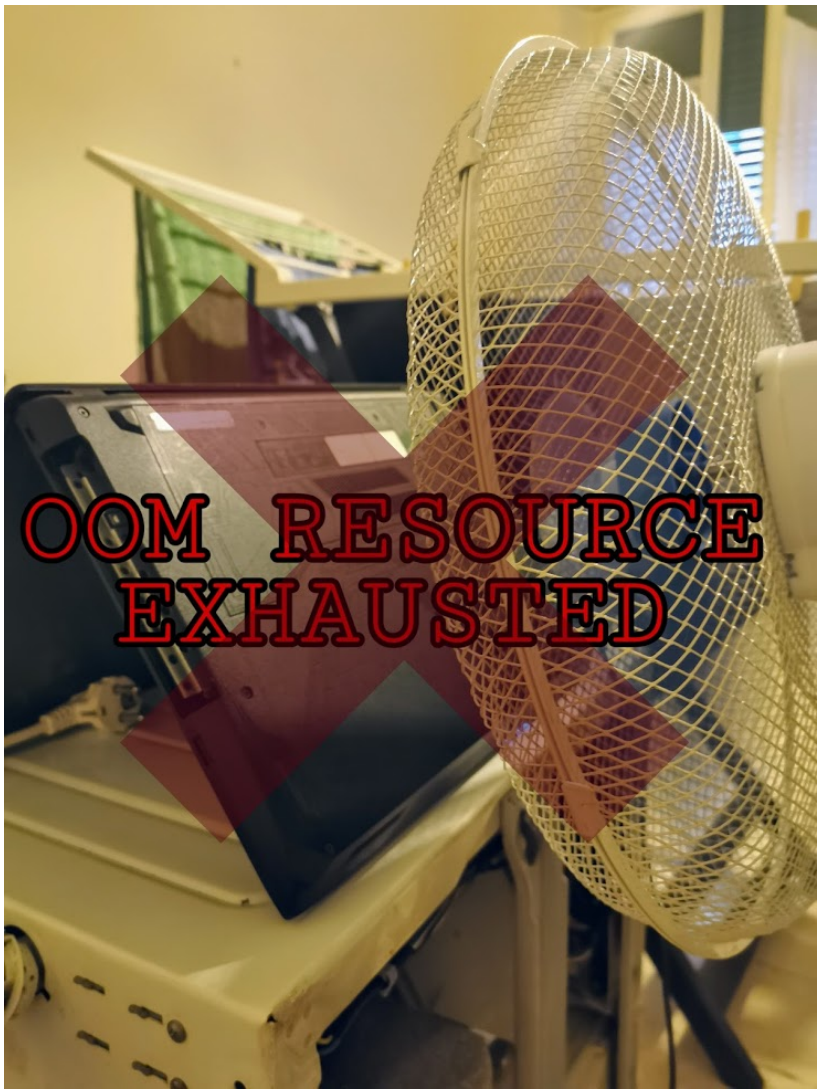
To obtain *smoothed images* from training, we apply the following steps:

- detect edge pixels using a standard Canny edge detector;
- dilate the edge regions;
- apply a Gaussian smoothing in the dilated edge regions.

Contents

Laptop

ASUS ROG Strix GL553VD Gaming Laptop with an NVIDIA GeForce GT 1050 4GB graphic card.

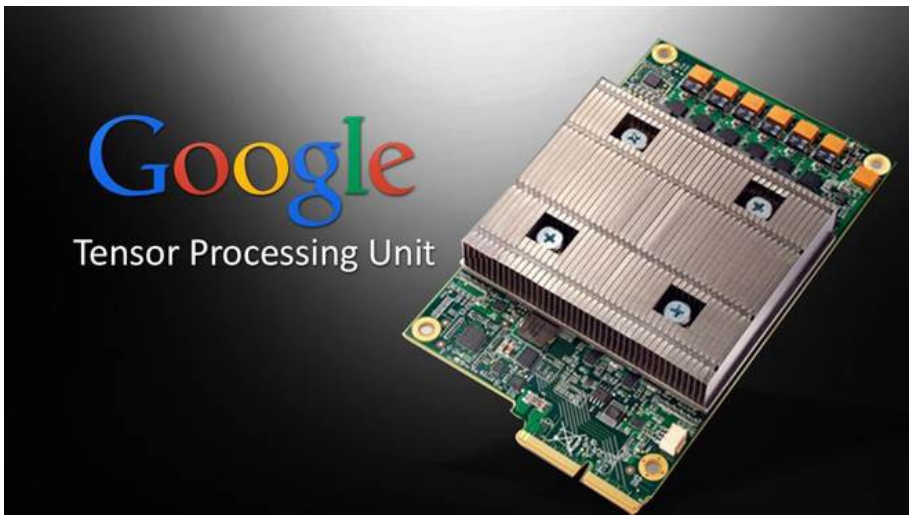


Laptop

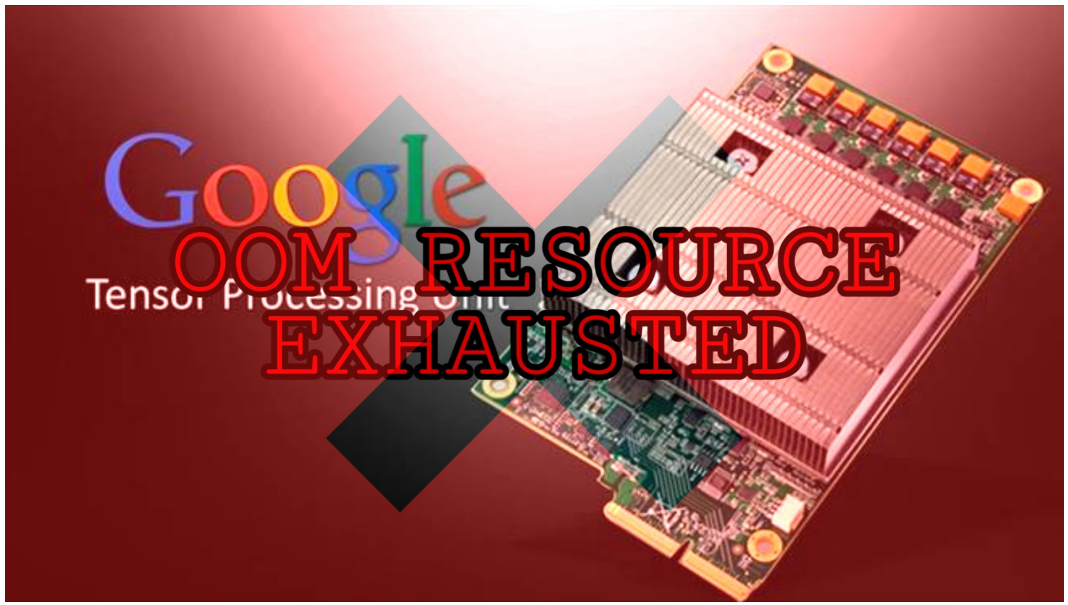
The training process fails during the pretraining process (only generator network working) BEFORE the end of the first epoch. We reduced the batch size to 1, without success. We thought: *"Ok, maybe the model is too heavy, the original experiments were done with a NVIDIA XP GPU graphic card, which has 12 GB of memory, 3x larger than our graphic card."*.

Colab - TPU

TPU's have been recently added to the Google Colab portfolio. The TPUv2 available from within Google Colab comes with a whopping 180 TFlops. It also comes with 64 GB High Bandwidth Memory (HBM).



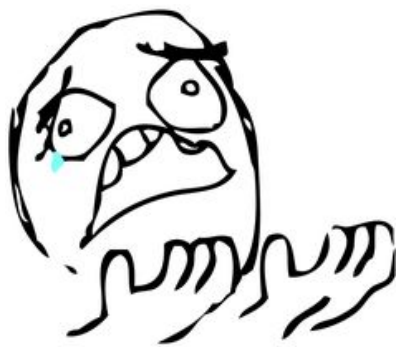
Colab - TPU



Also, the training was slower than local GPU, and we discovered that TPUs performance is worse compared to GPUs when batch size is low.

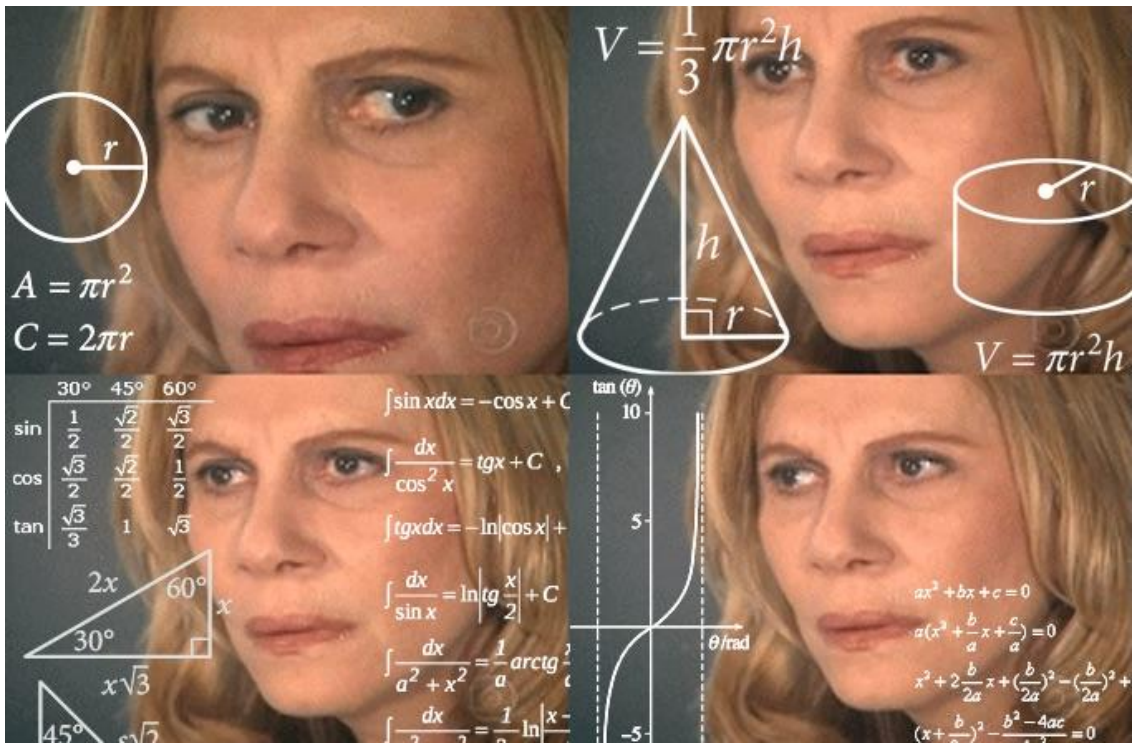
Colab - GPU

In the runtime of Google Colab we switched to GPU usage; the training was 10x faster w.r.t. TPU (1h for epoch vs 10h) but at the end of the first epoch we encountered again our known **Out Of Memory** error.



Idea

Then, an obvious fact:
“What if we save the model before the end of the epoch?”



Contents

Idea

Of course the training will stop with OOM error, but at least we can retrain from the last saved checkpoint!

Problems

But the entire training process is **OBNOXIOUS**:

- we have to run the training cell every epoch that lasts roughly 45 minutes.
- We have to keep the browser's window open, otherwise Google Colab resets the environment every hour of non-activity (so this means that every 45 minutes we have to check the activity state).
- Even if the running process is active Google Colab resets the environment every 12 hours.

Further Investigation

But what really causes the OOM error? If the available memory was really a problem, the training would not have ended the epoch.

Of course there is a memory leak somewhere in the code.

Stacktrace

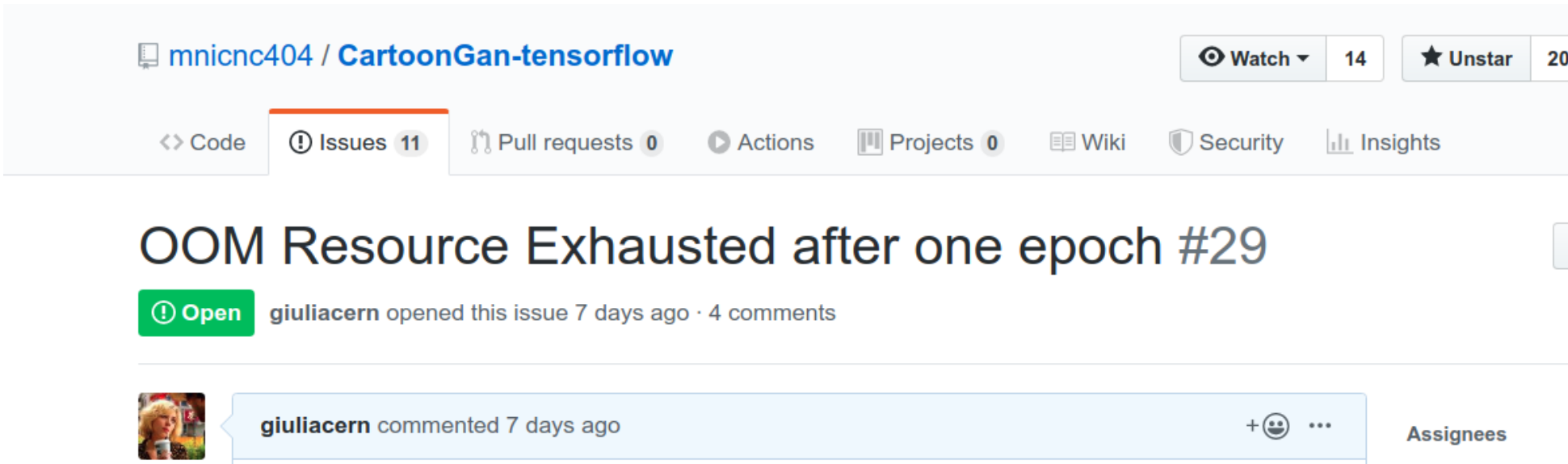
We looked better at the Stacktrace and we found the block of code that caused the OOM error.

```
with summary_writer.as_default():
    if not self.disable_sampling:
        val_fake_batch = tf.cast(
            (generator(val_real_batch, training=False) + 1) * 127.5, tf.uint8)
        img = np.expand_dims(self._save_generated_images(
                val_fake_batch,
                image_name=("gan_val_generated_images_at_epoch_"
                            f"{epoch_idx}_step_{step}.png")),
                0,
        )
        tf.summary.image('gan_val_generated_images', img, step=epoch)
```

Further Investigation

Appereantly, if we comment the previous block the training proceeds without errors!

We have opened an issue on Github, hoping that developers will take in consideration this problem.



Contents

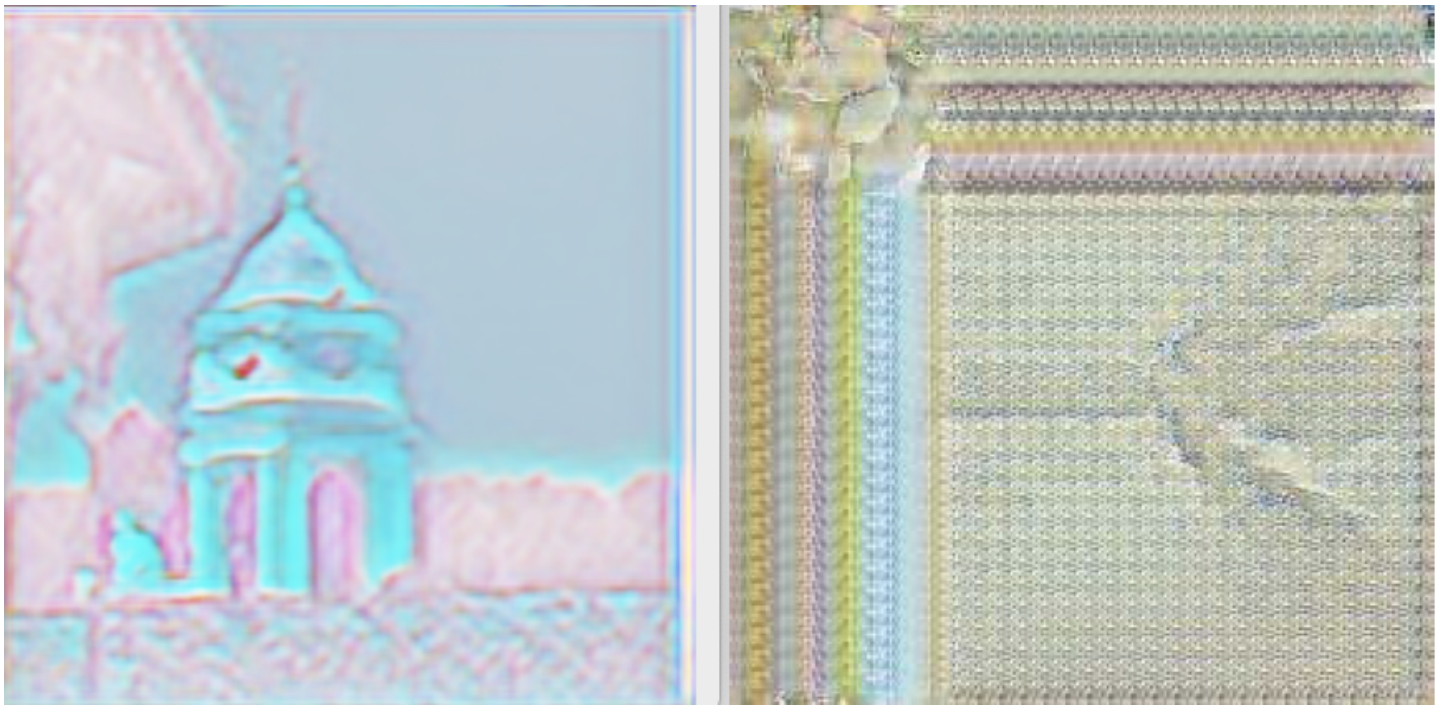
Train.py and structure

- Import Dataset and Image processing
- Main loop: speech between G and D (real or fake?)
- Adam Optimizer
- Save checkpoint and results
- Loss generation

Finally we compute the total loss function: $L_a + L_c * w$

Experiments

- The model has an initialization phase which aims to improve convergence
- Through the pretrain epoch the network will lead towards very fast results (keeping the input content)



results after a few epochs with and without initialization

Experiments

Hyperparameters:

```
--batch_size 1 \
--pretrain_epochs 1 \
--pretrain_learning_rate 2e-4 \
--content_lambda .4 \
--g_adv_lambda 8. \
--generator_lr 8e-5 \
--discriminator_lr 3e-5 \
--style_lambda 25. \
--dataset_name elective_data
```

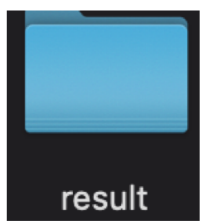
33 epochs trained, each one started manually from the previous checkpoint. Total training time: about 6 days and 9h.

Contents

First results

Results are immediately visibile epoch by epoch

Edge detection during first steps in training



Towards the results analysis

Thanks to the out.event files in Runs folder we can use

and with the command line **tensorboard –logdir runs**

we can analyze the training trend:

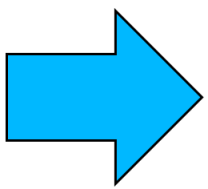
-output images
-loss functions

Results: output images

Input real images

+

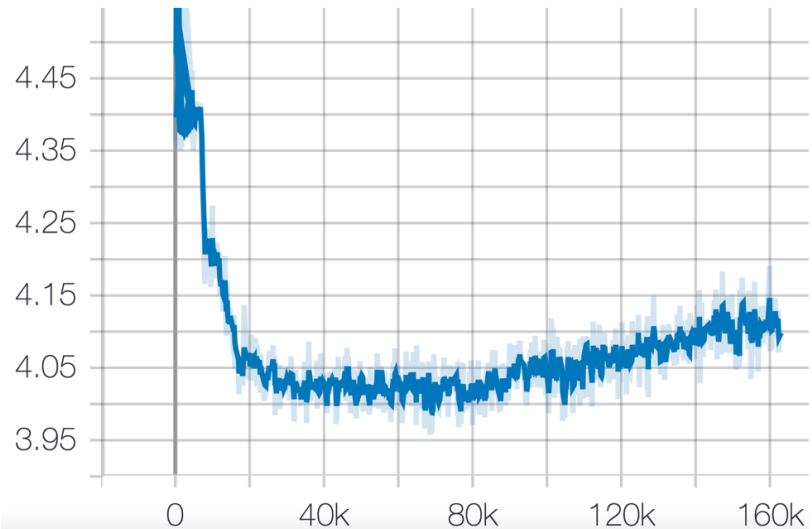
From input cartoon images



Output images after 30 epochs:

Results: loss functions

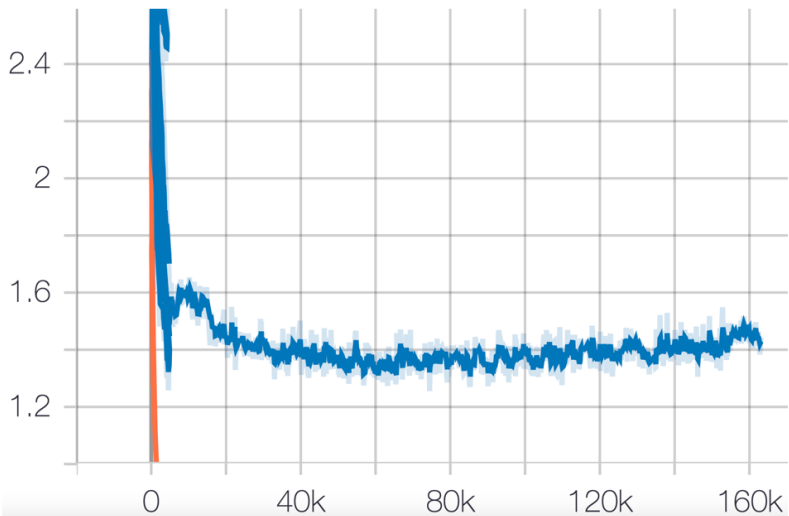
ADVERSARIAL LOSS



Final value: 4.09

State of art value: 2.6

CONTENT LOSS



Final value: 1.4

State of art value: 1.7

Contents

Conclusion

- GANs are a very powerfull tool in deep learning tasks...
- ...but pay attention to their instability!
- Future developments: what if with human faces?

THANKS FOR YOUR ATTENTION!